Gymnázium, Praha 6, Arabská 14

Obor programování

ROČNÍKOVÝ PROJEKT

Vykreslování grafů matematických funkcí

Vypracoval: Havránek Kryštof 1.E
Vedoucí práce: Ing. Daniel Kahoun

Květen 2019

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... Podpis autora

Poděkování Děkuji učiteli Danielu Kahounovi za pomoc při vypracovávání práce a svým předchozím učitelům, Martinu Horskému a Davidu Husníkovi, za naučení základů Javy a JavyFX ještě před nástupem na Gymnázium Arabská. Také děkuji Univerzitě Karlově, za poskytnutí šablony na práci ve formátu LaTeX [4].

Název práce: Vykreslování grafů matematických funkcí

Autor: Havránek Kryštof 1.E

Abstrakt: Cílem práce je vytvořit program schopný zobrazovat grafy matematických funkcí s jednou proměnou. Měl by podporovat základní matematické operace jako sčítání, odčítání, násobení, dělení, mocniny, goniometrické funkce a další funkce.

Title: Drawing graphs of mathematical functions

Authors: Havránek Kryštof 1.E

Abstract: Goal of work is to develop program capable of displaynig graphs of mathematical functions with one variable. It should also support basic mathematical operation like addition, substraction, multiplication, powers, trigonometric and other functions.

Titlel: Zeichnen von Graphen mathematischer Funktionen

Autoren: Havránek Kryštof 1.E

Abstrakt: Ziel dieser Werk ist ein Programm zu erstellen, das Graphen mathematischer Funktionen mit einer Variablen zeigt. Es sollte grundlegende mathematische Operationen wie Addition, Subtraktion, Multiplikation, Division, trigonometrische Funktionen und andere Funktionen unterstützen.

Obsah

Zadání práce a úvod

Předmětem práce je program, jenž v jednoduchém uživatelském prostředí umožní uživateli zobrazit grafy různých matematických funkcí s jednou neznámou. Program dokáže zobrazit hned několik grafů na jednou a vykreslené grafy může uložit do souboru jako obrázek. Uživatel má také možnost graf lobovolně přiblížit, nebo oddálit. Lze také nastavit libovolnou barvu grafu, pokud jí uživatel nezvolí použije se jedna z 10 předefinovaných pastelových barev. Společně s grafem funkce se také vykresluje měřítko obou os, to je stejné pro obě osy. Měřítko se mění automaticky společně s přiblížením grafu.

Program je napsaný v jazyce Java a na grafické zobrazení používá knihovnu JavaFX, při exportování a importování do/z souboru je použit JSON, s ním se operuje prostřednictvím knihovny json. Bonus jsem při vypracovávání práce stanovené neměl, funkce jsem přidával v pořadí jak mě napadly.

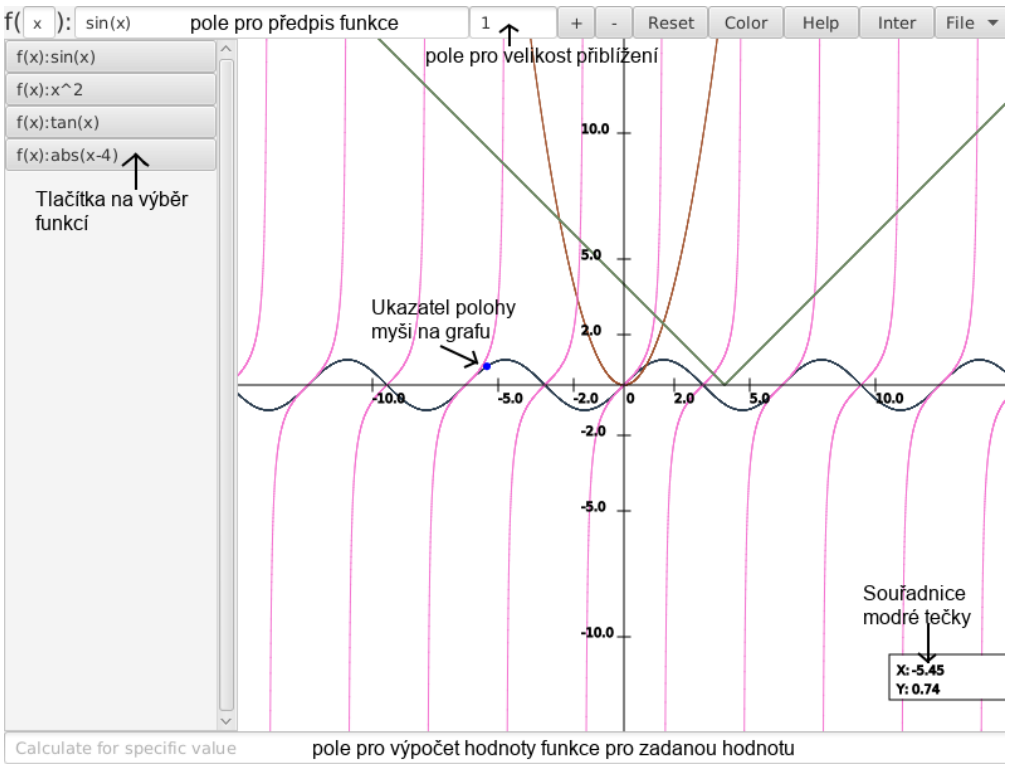
1. Struktura programu, popis částí programu

```
GraphDraw
├── FXMLDocument.fxml
├── FXMLDocumentController.java
├── GraphGraw.java
├── ParsedExpression.java
└── PosfixExpressionClac
    ├── OperatorStack.java
    ├── PostFixExpressionCalc.java
    └── Precedence.java
```

Program se skládá ze dvou částí, první je obsah složky PosthixExpressionCalc. Třída PostfixExpressionCalc, se stará o vypočítání hodnot funkce, či nalezení průsečíků dvou funkcí. Další dvě třídy, obsažené ve složce, jsou třídy pomocné, více pozornosti jim bude věnováno v kapitole o samotném backendu. Druhou částí programu je soubor tříd obsahující FXMLDocumentControler, GraphDraw a ParsedExpressions, soubor FXMLDocument.fxml složí k uložení rozložení programu. Třída FXMLDocumentControler spojuje grafickou část programu s jejím backendem, stará se o funkce tlačítek, nakonfigurování aplikace a vykreslení samotného grafu. Třída GraphDraw obsahuje metodu main, ta jen vytvoří a zobrazí scénu. Poslední třída – ParsedExpressions, slouží k uchování informací o všech aktuálně zobrazených funkcích, má také metody na importování/exportování z/do daného souboru.

2. UI programu / Frontend

2.1 Vizuální vzhled programu a ovládání



Okno programu

Ovládání programu je relativně jednoduché. Uživatel nejprve zadá proměnou do malého vstupního pole v horním rohu, tou může být jakékoliv písmeno v anglické abecedě, až na písmeno e, to slouží jako konstanta e, $e \approx 2.71$. Dále uživatel zadá předpis samotné funkce do pole napravo od pole s proměnou. Předpis je zapsán pomocí standardního matematického zápisu, program podporuje i některé matematické funkce, jedná se o absolutní hodnotu, goniometrické funkce a jejich opaky, funkce ceil, floor, sqrt a nalezení menšího, či většího čísla ze dvou zadaných. Je také možné použití konstant, těmi jsou e, π, ϕ. Nutno upozornit, že desetinná čísla jsou oddělena desetinou tečkou, nikoliv čárkou. Pro vykreslení grafu stačí v poli s předpisem stisknout klávesu enter. Před vykreslením, je ještě možnost nastavit barvu grafu funkce, ty se standardně nastavují dle předdefinovaných barev, pomocí tlačítka color si však uživatel může nastavit vlastní. Barvu grafu lze po vykreslení jednoduše změní tím, že nechá vykreslit graf znovu s jinou barvou.

Pokud bude uživatel přejíždět myší po plátně, tak na grafu se bude zobrazovat malá modrá tečka, její pozice je zobrazena ve spodním pravém rohu. Lze také vypočítal hodnotu grafu pro určenou hodnotu neznámé, k tomu slouží pole vespodu okna. Mezi grafy, pro které se počítají hodnoty se přechází prostřednictvím, menu na levé strany obrazovky, do něhož se graf automaticky po zobrazení přidá, pokud menu obsahuje mnoho grafů, stačí použít posuvník a stáhnout menu dolů. Vykreslené grafy si uživatel může libovolně oddalovat a přibližovat, za pomocí scrollování na myši, zadávacího pole na horní liště, či tlačítek + a -, jež najde také na horní liště. Maximální hodnota je přiblížení je 1 a minimální 100.

Pokud si uživatel chce nakreslené grafy uchovat do příštího spuštění programu (program nic dlouhodobě neukládá do paměti), může z menu File vybrat možnost Export. Ta uživateli zobrazí průzkumník souborů, kde vybere, či vytvoří soubor do něhož se uloží informace o vykresleních

grafech. Po další spuštění pak jednoduše zvolí v menu File položkou Import a vybere soubor, který před ukončením nechal vytvořit. Pokud uživatel chce uložit vykreslené grafy jen jako obrázek, pak z menu File vybere možnost Save Img.

Program má ještě možnost najít souřadnice průsečíků dvou grafů, které leží v rozmezí maxima a minima osy x na plátně. K tomu slouží tlačítko Inter (z anglického intersection), po jeho stisknutí pomocí menu nakreslených funkcí vybereme jednu z funkcí, jako druhá funkce slouží ta, která byla aktuálně vybraná před přechodem do Inter módu. Pokud uživatel vybere funkci, která je stejná jako funkce, která byla vybrána před přechodem do módu Inter, tak program zobrazí body kde funkce protíná osu X. Pro vrácení zpět do módu vykreslování je třeba stisknout tlačítko Back.

Nápovědu k použití programu lze zobrazit lehce pomocí tlačítka Help, nutno zdůraznit, že stejně jako zbytek programu je v jazyce anglickém.

2.2 Programová stránka UI

Program je psaný za pomocí JavaFX konkrétněji v JavaFXML, jedná se proto o GUI aplikaci vykreslující se v okně. K jejímu spuštění je tak k zapotřebí kromě Javy i nějaký systém pro vytváření oken jako X.org. Pevné rozložení prvků na obrazovce je uloženo v souboru s příponou .fxml, dynamické prvky UI jsou pak definovány v samotném programu a to ve dvou případech. Zaprvé se jedná o tlačítka na postranní liště. Zadruhé je jím mód pro zobrazení souřadnic průsečíků dvou grafů, jelikož se jedná jen o jeden prvek UI, bylo zbytečné vytvářet novou scénu s vlastním .fxml souborem. Samotná plocha na vykreslování grafů, se také mění dynamicky, to je něco co Canvas v základu neumí, požadovaného výsledku bylo dosaženo použitím objektu Pane. Jeho velikost se již dynamiky mění s rozměry okna, na Pane se dále zavolá metoda getChildrem kde se mu přidá Canvas. Tomu se pak svážou rozměry s rozměry Pane a přidají Listenery na rozměry okna, ty v případě změny graf překreslí. K tomuto účelu slouží třída ResizableCanvas, ta se nachází v souboru FXMLDocumentControler. Společně s velikostí okna pro vykreslení se mění i velikost polí na zadávání předpisu funkce a výpočtu hodnoty funkce pro libovolnou hodnotu. Poznámka autora: Třída ResizableCanvas byla převzata z mého předchozího projektu, ten jsem programoval na kroužku před nástupem na tuto školu, neuvádím proto citaci i když kód je podobný s tím co uvádí na internetu.

Základem rozložení programu je BorderPane, v jeho horním kontejneru se nachází HBox. Ten obsahuje většinu ovládacích prvků programu – pole pro zadání proměnné (to má po stranách dva Labely s texty f(a)=), pole pro zadání předpisu funkce pole zadání hodnoty přiblížení grafu. Dále tlačítka na zmenšení/zvětšení přiblížení, zresetování programu do původního stavu, nastavení barvy grafu (před jeho vykreslením), zobrazení nápovědy, přejití do módu pro hledání průsečíků a menu pro operace se soubory – export do obrázku, export dat z ParsedExpression do JSON a import dat ze souboru JSON. Ve středu BorderPane je Pane (se kterým je svázaný Canvas). V levém kontejneru se dále nachází ScrollPane, do kterého je zabalen VBox. Na závěr ve spodním kontejneru se nachází TextField, pro výpočet hodnoty funkce pro danou hodnotu. V módu na zobrazení průsečíků se místo objektu Pane nastaví na střed TextArea, zároveň se také nastaví globální proměnná ovlivňující funkci tlačítek na vybrání funkce. Ty tak místo nastavení proměnných do TextFieldů a kalkulátoru postfixových výrazů, zavolají funkci v PostFixExpressionCalculator. Samozřejmě se také změní text tlačítka Inter na text Back, tlačítko potom složí k návratu do normálního módu. Dle původního plánu by tlačítko Inter měnilo funkci tlačítek z menu, ty by jim byly přiřazeny jakožto funkce lamba, z toho ale sešlo, protože nejde měnit funkci tlačítka na funkci, která ještě nebyla definovaná, proto bylo také nutno přidat další globální proměnou.

3. Backend

3.1 Postfixový kalkulátor

3.1.1 Dev log

Nejtěžším problémem práce bylo vypočítání hodnoty funkce. Java totiž, podobně jako většina hlavních programovacích jazyků, neumožňuje vypočítat hodnotu matematického výrazu, který je zadán v podobě Stringu.

Prvotním plánem bylo použít objekt zvaný ScriptEngineManager, jemuž se přidá skrtiptovací engine, jako je třeba JavaScript a ten již dokáže vypočítat hodnotu výrazu z neznámé. S použitím ScriptEngineManageru by pak stačilo převést uživatelem zadaný výraz do zápisu s voláním na knihovnu Math, čehož lze lehce dosáhnout pomocí jednoho cyklu a podcyklu pokud obsahuje mocninu, či odmocninu, kde je potřeba najít základ a mocnitele. Toto řešení má ale dva hlavní problémy. Zaprvé umožňuje spuštění JavaScriptového kódu, který zadá uživatel do pole pro předpis funkce, to sice není velký problém v tomto případě, ale i tak je lepší, aby uživatel neměl možnost spouštět příkazy tam kde být spuštěny nemají. Větším problémem byla rychlost výpočtu hodnot funkce, trvalo několik sekund, než se vypočítaly hodnoty jedné funkce. I tak však program nebyl funkční, u funkcí jako je tan, kde se na malém úseku přejde z velkého čísla do malého čísla se nevykreslovaly velké úseky, a nebylo možno vypočítat dostatek hodnot v přijatelném čase. Toto je problém, který se objevoval pořád během vývojového cyklu. Použití ScriptEngineManageru tedy nepřicházelo v úvahu, jako další možnost se naskytlo rekurzivně vypočítat rovnici sám.

U této varianty byly dvě možnosti, a to vytvořit objektovou reprezentaci funkce, či pokaždé vypočítat samotný String. Objektová reprezentace vyžadovala udělat node pro každou specifickou část (znaménko, číslo, proměnná, funkce), pomocí nich pak vytvořit strom a ten zespodu vždy vypočítat. Toto řešení bylo velmi dlouhé a pochybně efektivní. Způsob vypočítat pokaždé String byl zase pomalý, protože se musel string vždy zpracovávat. Kód byl také velmi nepřehledný a značně inspirovaný internetem.

Jako další možnost se jevilo použití jiného matematického zápisu než je ten standardní, infixový. Ten by totiž umožňoval lehčí výpočet funkce, zatímco standardní zápis obsahuje složité konstrukce jako závorky, či přednost početních operací, tak například postfixový je nemá. V postfixové zápisu je tedy mnohem lehčí určit hodnotu výrazu. Například příklad $(5+1)*(6-3)$, je v postfixovém zápise $51+63-*$, tento výraz již lze jednoduše vypočítat, za pomocí zásobníku, kam se budou ukládat čísla a narazí li se na nějakou operaci, jednoduše si program za zásobníku vyhodí číslo/a provede na nich operaci. Výsledek poté uloží zpět do zásobníku, nakonec v zásobníku zůstane jen jedno číslo a to je konečným výsledkem. Způsob výpočtu přes postfix je navíc relativně rychlý, protože složitost výpočtu je pouhý O(n).

3.1.2 Gramatika uživatelova zápisu

Matematický zápis předpisu funkce se vesměs řídí standardními matematickými pravidly. Desetinná čísla jsou však zapsána s desetinou tečkou, aby byl zápis konzistentní se zbytkem programu, který je v angličtině. Proměnou může být jakékoliv písmeno z anglické abecedy, až na písmeno e. To a řetězce phi a pi jsou vyhrazeny pro matematické konstanty. Standardní operace se zapisují pomocí symbolů +, -, *, /. Předpis může či nemusí obsahovat mezery. K mocnině slouží značka ^, pro zapsání odmocniny se do závorky zapíše převrácená hodnota mocniny, např. $\sqrt{3} = 3\hat{}(1/2)$. Funkce jsou napsané standardně a jejich argumenty jsou v závorkách, např sin(x), tan(4). Absolutní hodnoty se zapisují pomocí funkce abs, znak | nebude zpracován a vyústí v chybovou hlášku. Program podporuje dvě funkce (max a min), které potřebují dva argumenty, v jejich případě budou argumenty odděleny čárkou. Chce li uživatel použít záporné číslo např $x * (-3)$, tak číslo musí být v závorkách. Jen v případě, že se jedná o první znak zápisu může být bez závorek. Bude li číslo před závorkou, nebo před funkcí, program vyhodí chybovou hlášku, závorka nebude vynásobena.

3.1.3 Počítání s postfixem

Převedení do postfixového zápisu

Na výpočet s postfixovým zápisem je potřeba nejprve převést uživatelem zadaný infixový String do postfixového Listu (s listem se lehčeji zachází). K tomuto účelu slouží algoritmus Shunting-yard vymyšlený Edsgerem Dijkstrou [1]. Ten funguje na jednoduchém principu jednoho zásobníku a listu. Nalezne li algoritmus číslo, neznámou, či proměnou přidá ji na konec listu. Nalezna li operaci, tak začne procházet zásobník a přidají se do výsledného liste všechny operace, které mají větší, nebo rovnou přednost. Takto bude čísla přidávat, a z zásobníku odstraňovat, dokud nenajde číslo s menší předností, levou závorku, funkci, nebo nedojde na konec zásobníku. Po skončení přidávání operaci přidá do zásobníku. Nalezne li levou závorku, či funkci jednoduše ji přidá do zásobníku. Nalezne li pravou závorku ze zásobníku postupně přidá vše dokud nenalezna závorku levou, tu smaže a nepřidá do listu. Nakonec postupně přidá celý zásobník na konec listu. V programu je tento algoritmus implementován pomocí tří tříd.

První je třída Precedence, ta obsahuje Enumy s informacemi o přednosti početních operací. Ta má jen jednu metodu s switch case, ta vrátí početní hodnotu operace podle zadaného Stringu.

Druhou třídou je OperatorStack, ta reprezentuje zásobník s početními operacemi. Obsahuje metody na standardní obsluhu zásobníku, přidání operace, funkce, metodu k zavolání při nalezení závorky. Zároveň může zobrazit Alert, pokud nebudou závorky v páru.

Třetí a hlavní složkou je třída PostfixExpressionCalc. O převedení do postfixu se stará metoda parse, ta je automaticky zavolána konstruktorem, konkrétně tím, který jako argument bere dva Stringy. Metoda funguje na jednoduchém principu, kdy postupně prochází String a hledá změny ve znacích, proto musí na začátku ke Stringu přičíst mezeru. U změn si vždy pamatuje indexy a pomocí nich String dělí na substringy. Těmi už jsou samotná čísla, konstanty, funkce. Jeli řetězec číslo jednoduše ho přidá do Listu, obsahuje li znaky, zjistí zda se jedná o proměnou, konstantu či funkci. Při každé iteraci proběhne ještě switch case, který se řídí aktuálním charakterem, ten přidává operace +, -, *, / do zásobníku a zajistí správné zpracování závorek a čárky. U závorek zkontroluje zda je mezi nimi jen jedno číslo či neznámá, pokud ano, tak obsah mezi závorkami přidá do výsledného listu. Na závěr metoda vyprázdní zásobník s operacemi do listu. Dojde li někde během převádění k chybě (uživatelův vstup je v rozporu se stanovenou gramatikou), tak program vytvoří okno s upozorněním a nastaví proměnou isExpressionCalculable na false. Program pak ví, že nemá smysl se snažit příklad vypočítat. Po skončení se zavolá metoda setUpRecognitionArray, ta nastaví pole, které obsahuje informace o typu jednotlivých prvků postfixového zápisu, zda se jedná o číslo, proměnou, atd. V praxi to vede ke zrychlení výpočtu, neb není potřeba při každém výpočtu zjišťovat typ prvku.

Výpočet postfixového zápisu

K výpočtu postfixového zápisu slouží metoda evaluateExpression. Výpočet je založen na jednom cyklu, který postupně prochází pole a pomocí switche buďto přidá do zásobníku, nebo si z zásobníku vezme prvky a provede na nich operaci. Zároveň během přidávání do zásobníku se za neznámou dosadí číslo, které metoda dostane v argumentu. Pokud najde funkci, kterou nezná vytvoří Alert a nastaví proměnou isExpressionCalculable na false. Pokud dojde k takové chybě či jiné chybě během výpočtu, program vrátí Double.NaN.

Hledání průsečíků

Třída PostfixExpressionCalc obsahuje také metodu na nalezení průsečíků. Ta funguje na relativně jednoduchém principu, metoda dostane jako argument druhou rovnici a proměnou, kterou používá. Pokud je proměnná jiná než, kterou používá funkce, která je aktuálně nastavená v PostFixExpressionCalc, tak proběhne cyklus, který proměnné srovná. Rovnice poté od sebe odečte, v postfixovém zápisu je nejprve třeba přidat prvky druhé rovnice, až potom přidat znaménko mínus. Po těchto úpravách můžeme již začít hledat pro jaké hodnoty se výraz rovná nule. Na hledání těchto bodů je použita metoda zvaná bisection method [2]. Ta je založená na principu, že pokud máme hodnotu funkce, která je kladná, a hodnotu funkce, která je záporná, nutně se mezi nimi nachází průsečík s nulou. Nejprve tedy program hrubě počítá hodnoty funkce

a najde li nějaké zlomové hodnoty, kde se mění znaménko, pomocí binary search nalezne přesný bod. Program, hledá jen v rozmezí osy x, která je právě zobrazená na grafu.

Je pravda, že jiné algoritmy na řešení rovnic sice najdou řešení rovnic i bez předem zadaného rozmezí, ty však často vyžadují derivaci funkce. Vytváření derivací se však prvky jako min a max značně komplikuje. Navíc mnohé z nich udají jen jeden výsledek rovnice. Proto byla použita tato metoda, ačkoliv se jedná neefektivní řešení. Co se týče rychlosti se však, díky jinak rychlému počítání rovnice, uživatel nemusí na výsledek dlouho čekat. Metoda po skončení vrátí List s body na souřadnici x, kde dochází k průniku.

Existují také vstupy pro které nemusí hrubé prohledávání najít zlomové body, tyto možnosti program neřeší. Příkladem může být f(x)=x^2 - 0.000 001.

3.2 Controller k FXML souboru

3.2.1 Inicizlizace programu

O samotné řízení programu se stará třída FXMLDocumentController. Scénu a Stage však nejdříve vytvoří třída GraphDraw, ta taky zavolá metodu setStage v FXMLDocumentConroller, která se postará o inicializaci UI, aby byl program připraven pro uživatelův vstup. V metodě se vytvoří objekt ResizableCanvas, a nastaví se globální proměnné jako je GraphicsContext2D, či CustomColorDialog. Před skončením se zavolají metody drawScale a drawMouseBox, které na plátno nakreslí osy a ohraničení pro zobrazení pozice modrého bodu. Je sporná užitelnost jejich použití, záleží na tom, jak který desktop, či window manager zachází s novými okny. Například v případě window manageru i3wm se velikost okna několikrát změní ještě před tím, než se inicializuje GraphicsContext, což vede k nullPointerExeption (všechny metody s vykreslováním jsou proti ní chráněni), ale také už zavolají překreslení Listenery v ResizableCanvas, čili se ohrádka a osy vykreslí několikrát. Vykreslování se obecně chová zvláštně v této fázi, varianta uvedená v programu však funguje.

3.2.2 Metody sloužící k vykreslování na Canvas

Třída obsahuje celkem 8 metod, které vykreslují na Canvas, či volají metody vykreslující na Canvas. Nejméně zajímavou z nich je metoda drawMouseBox, ta se stará čistě o vykreslení malého okénka pro zobrazení pozice modrého bodu. Je však samostatná kvůli přehlednosti kódu a navíc ji volá metoda (spíše lambda funkce) mouseMovedInCanas. Ta bude popsána o něco později. Další metodu je metoda drawScale, ta se stará o vykreslení os a jejich popisků. Měřítko osy je dáno hodnotou zoom, ta je v rozmezí od 19 do 119. Hraniční hodnoty jsou způsobeny funkcemi jako je tan, s tímto měřítkem na fullhd monitoru nebude chybět kus jejich grafu. Když se měřítko nastavovalo automaticky, aby se tomuto nedostatku předešlo, bylo zobrazování nepřehledné, proto je v konečné verzi omezené těmito hodnotami. I přesto, že funkce např. f(x)=$tan(x)/100$, nebude vykreslena správně. Metoda DrawScale také uloží obrázek s aktuálním stavem Canvasu do proměnné, metoda je totiž vždy volána před metodou drawMouseBox, jedná se tedy o stav kdy jsou na plátně jen grafy funkcí a osy. Obrázek je pak používán lamba funkcí mouseMovedInCavas, ta vždy na plátno nakreslí obrázek grafů na něj modrou tečku, ta se tak pohybuje plynule s pohybem myši, neboť se vždy nemusí překreslovat všechny grafy. Metoda reDrawFunctions pomocí cyklu prochází vše uložené v ParsedExpressions a každý graf nakreslí, volají ji metody menuImportAction, update a drawGrahAction. O samotné vykreslení grafu se stará metoda drawToCanas, ta dostane jako argument List obsahující Point2D, ty projde a spojí je čarami.

Metoda drawGraphAction

Tato metoda je volána vždy po zmáčknutí klávesy v poli pro předpis funkce, její obsah se však vykoná jen pokud je zmáčknuta klávesa enter. Pokud je stisknuta klávesa enter, vytvoří metoda objekt PostfixExpressionCalc a jako argumenty do konstruktoru dá infixový předpis funkce a její proměnou. Poté provede jeden kontrolní výpočet, zdaří li se uloží si do proměnné kopii postfixového předpisu. Než proběhne vykreslení zkontroluje zda uživatel změnil barvu funkce,

jestli ne tak nastaví jednu z předdefinovaných barev. Není li funkce uložená v ParsedExpression tak se uloží, pokud byla změněna barva, tak se přepíše na novou hodnotu. Zároveň se vytvoří tlačítko v HBoxu vlevo na vybrání funkce. Poté se pomocí metody reDrawFunction zobrazí se samotný graf.

3.2.3 Třída ParsedExpressions

Třída ParsedExpression slouží k uchování funkcí, které lze nakreslit. Parametry funkcí jsou uloženy ve 4 Listech, ty si pamatují infixový zápis, postfixový zápis, barvy grafu a proměnou. Na obsluhu obsahuje standardní metody, jako jsou getry a setry. Obsahuje i metody na generování a psaní do souboru – metody importFromJSON a exportToJSON. Metoda pro exportování funguje na primitivním principu – postupně prochází všechny Listy a data z nich zapisuje do JSON, z toho důvodu ani nebyly použity nástroje z knihovny json. Struktura souboru je jednoduchá, obsahuje pole functions, v něm jsou postupně vypsané funkce.

```
{
    "functions":[
        {
            "infix":"sin(x)",
            "postfix":"x, sin",
            "color":"#2c3e50",
            "variable":"x"
        },
        {
            "infix":"abs(abs(x+1)-abs(x-2))",
            "postfix":"x, 1, +, abs, x, 2, -, abs, -, abs",
            "color":"#a0522d",
            "variable":"x"
        }
    ]
}
```

Příklad JSON souboru

Při čtení JSON je použita knihovna json [3]. Obsah celého souboru se nejprve načte do jednoho Stringu. Ten je použit jako argument v konstruktoru pro JSONObject. Z toho se poté dostane pole functions, to se uloží do JSONArray. Program pak pole prochází zpracuje jejich vstup (postfix převede na List, k barvě přidá 0x a ff, atd.), provede kontrolní výpočet a pokud vše proběhne správně funkci přidá do Listů. Metoda vrátí true, jestliže něco bylo přidané do Listů. FXMLDocumentController v tomto případě totiž překreslí grafy.

Předpokládá se, že velikost souboru je malá, proto je předán JSON do JSONObject v podobě Stringu a na čtení souborů není vytvořeno není vlákno.

Ukládání rovnic by šlo řešit za pomocí ObservableList a objektu, který by reprezentoval jednu funkci. Třída FXMLDocumentControler, však potom musela obsahovat metody pro exportování a importování ze souboru. Tím se stávala (dle názoru autora) nepřehlednou. Byl proto použit způsob, kde jedna třída obsahuje čtyři Listy.

Závěr

Dle mého názoru byly cíle práce naplněny. Ve své finální podobě program kromě vykreslování grafů dokáže grafy i exportovat. Zároveň je také možno hledat dvou průsečíky grafů. Jediným výrazným nedostatkem je možné chybné zobrazení u funkcí jako je tan, zde, ale nevím jak problém vyřešit, protože nejde předpokládat, že graf půjde automaticky do nekonečna u všech takových případů. Ideální řešení by nejspíš vyžadovalo komplexnější analýzy předpisu funkce, což je něco co je nad mé schopnosti. Samozřejmě je to však jedna složka která je otevřená k dalšímu rozvinutí a vylepšení. Celá práce je dostupná na stránce GitHub pod licencí WTFPL (URL: `https://github.com/havrak/GraphDraw`).

Seznam použité literatury

[1] Dijkstra, E. W. (1961). Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60. Technical Report 35, Mathematisch Centrum, Amsterdam [online]. Dostupné z `http://www.cs.utexas.edu/users/EWD/MCReps/MR35.PDF`. [cit 2019-17-5].

[2] kolektiv (2019). *Bisection method*. Wikimedia Foundation [online]. Dostupné z `https://en.wikipedia.org/wiki/Bisection_method`. [cit 2019-17-5].

[3] Leary, S. (2015). *JSON-java*. GitHub repository [Online]. Dostupné z `https://github.com/charlespwd/project-title`. [cit 2019-17-5].

[4] Martin, M., Arnošt, K. a Michal, K. (2015). *Šablona LaTeX*. MFF UK [online]. Dostupné z `https://www.mff.cuni.cz/cs/studenti/bakalarske-studium/statni-zaverecne-zkousky/bakalarske-prace`. [cit 2019-17-5].